CMSC 491/691

Lecture 6

Image Features II

Last Lecture

- What is a feature ?
- Why are features useful ?
- What is a “good” feature ?
- How to detect features (Harris Corner Detector)

Recap: Harris Corner Detector

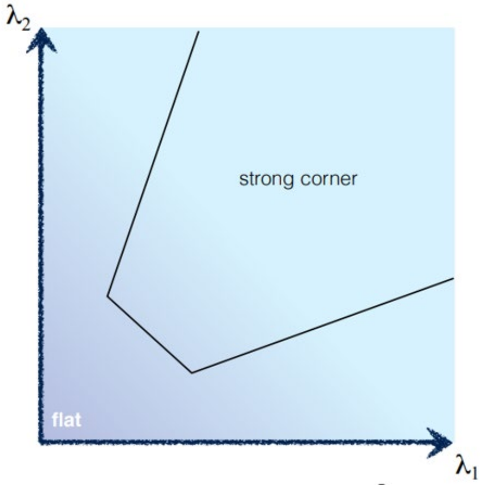
- **Key Idea:** Corners are good. Edges are OK. Flat regions are meh.
- Good feature ➔ high error:

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u,y+v) - I(x,y)]^2$$

  - But this is slow to compute
- Use Taylor Approximation:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

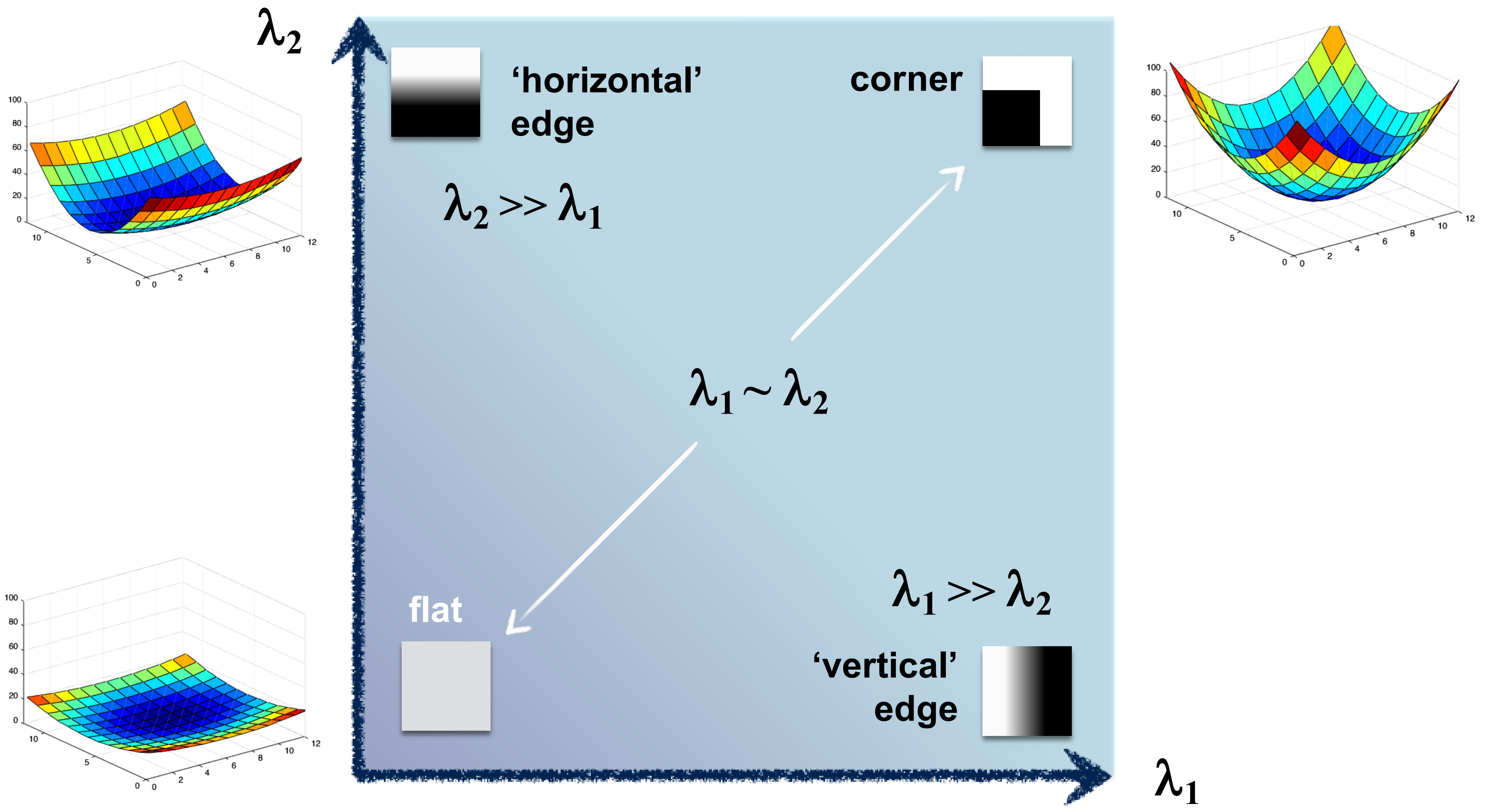
  - Use SVD on H
- Do thresholding in Eigenspace to select features



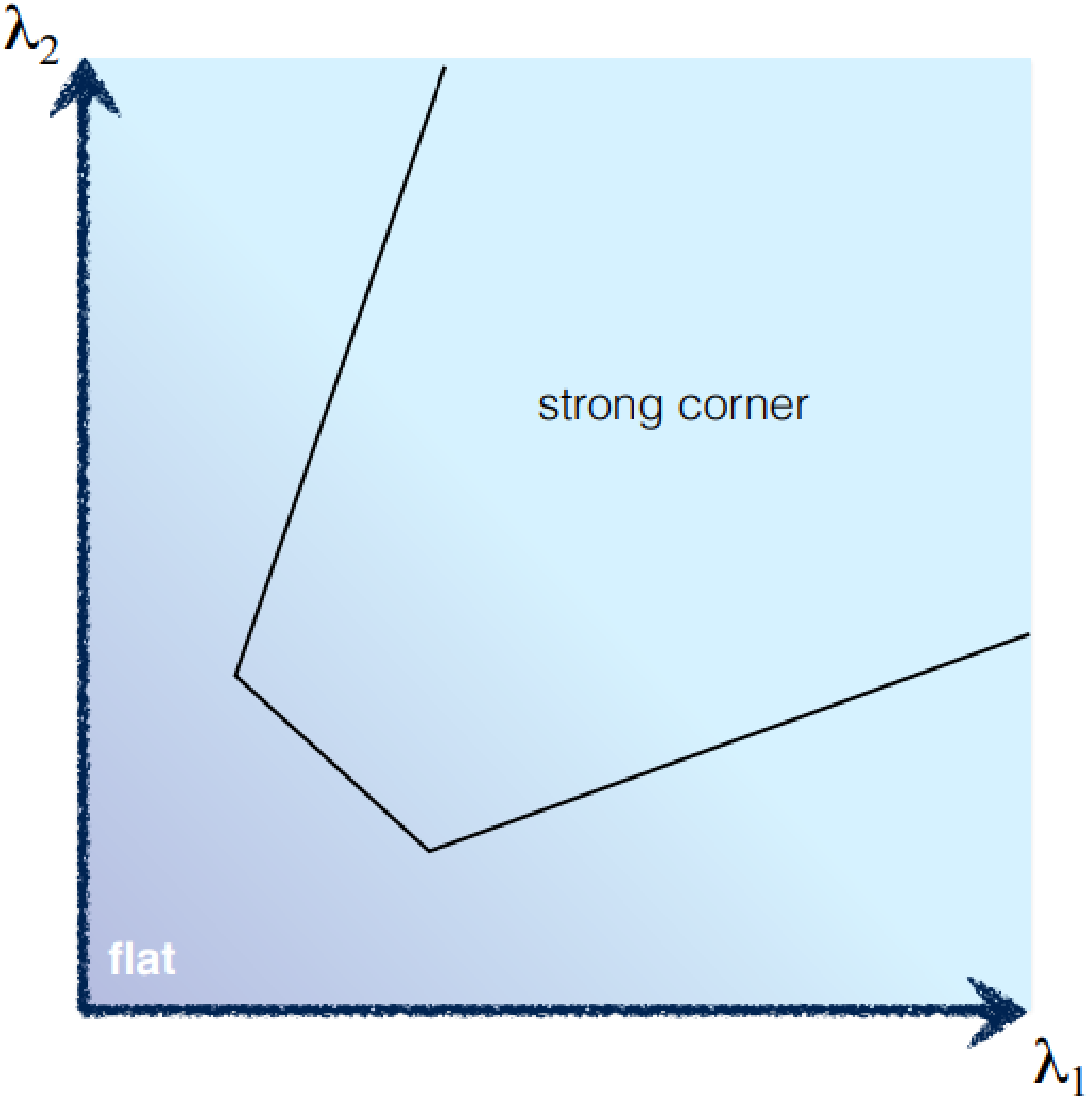
Recall …

Corner Detection in Eigenspace

interpreting eigenvalues



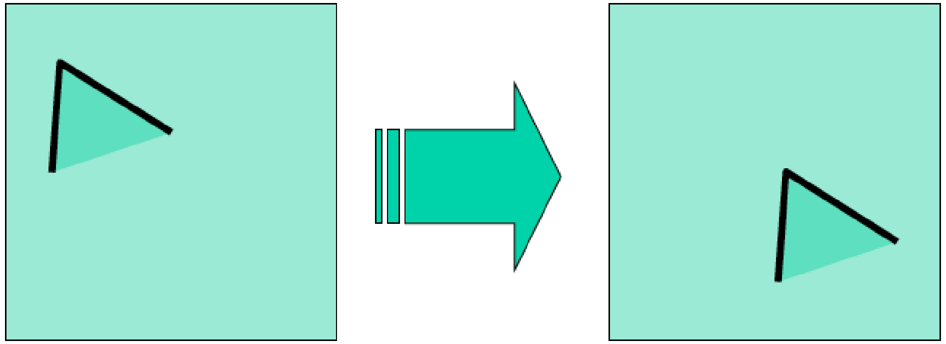
5. Use threshold on eigenvalues to detect corners



Think of a function to score 'cornerness'

Translation/Rotation Covariance

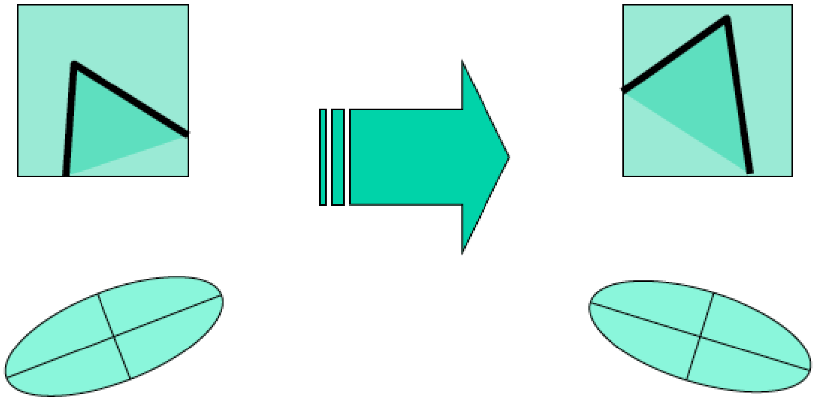
Image translation



- **Derivatives and window function are shift-invariant**

Corner location is covariant w.r.t. translation

Image rotation

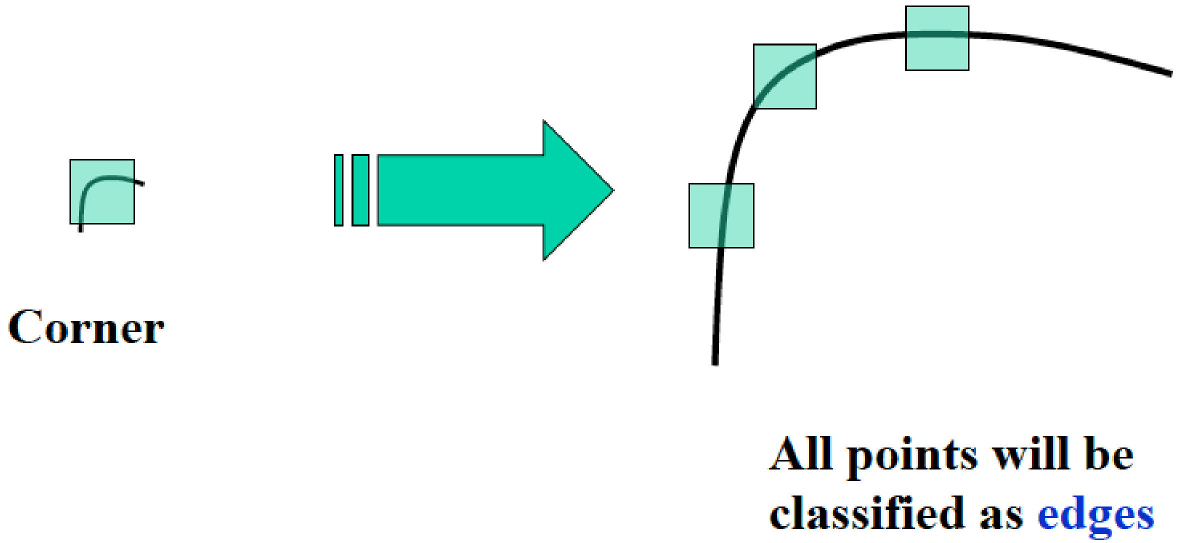


Second moment ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner location is covariant w.r.t. rotation

Scaling

Scaling



Corner location is not covariant to scaling!

How do we handle scale?

After feature detection, how do we match features in multiple images (feature description and matching)

Harris Corner Detector

Rotation invariant?

Scale invariant?

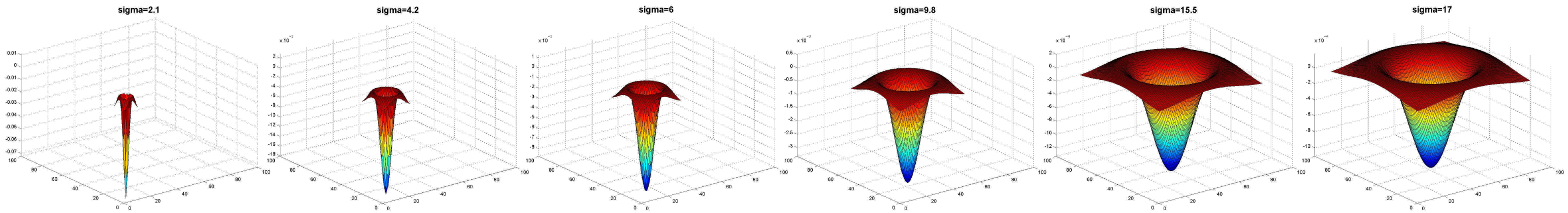
Harris Corner Detector

Rotation invariant?

Scale invariant?

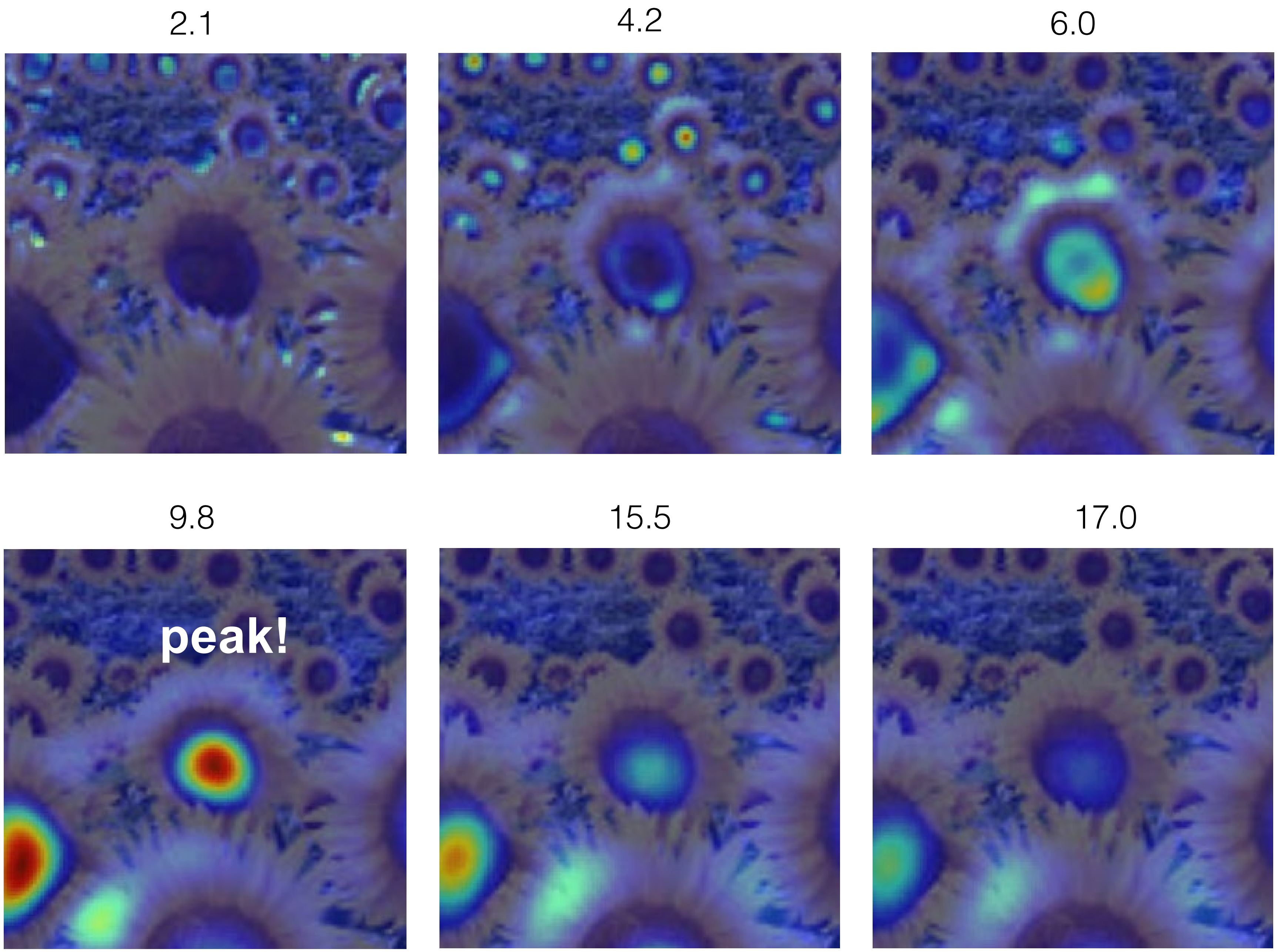
Multi-Scale 2D Blob Detector

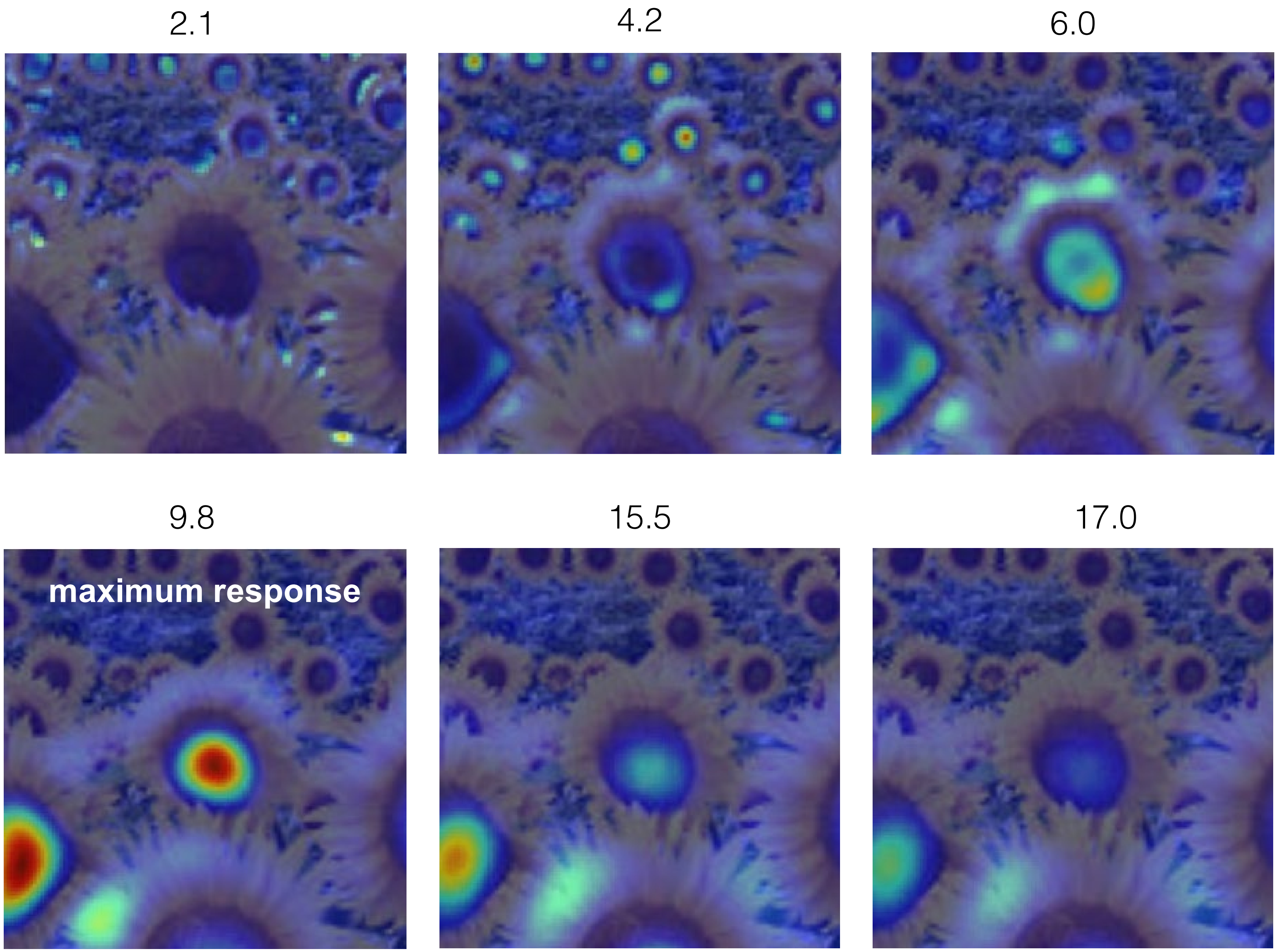
What happens if you apply different Laplacian filters?

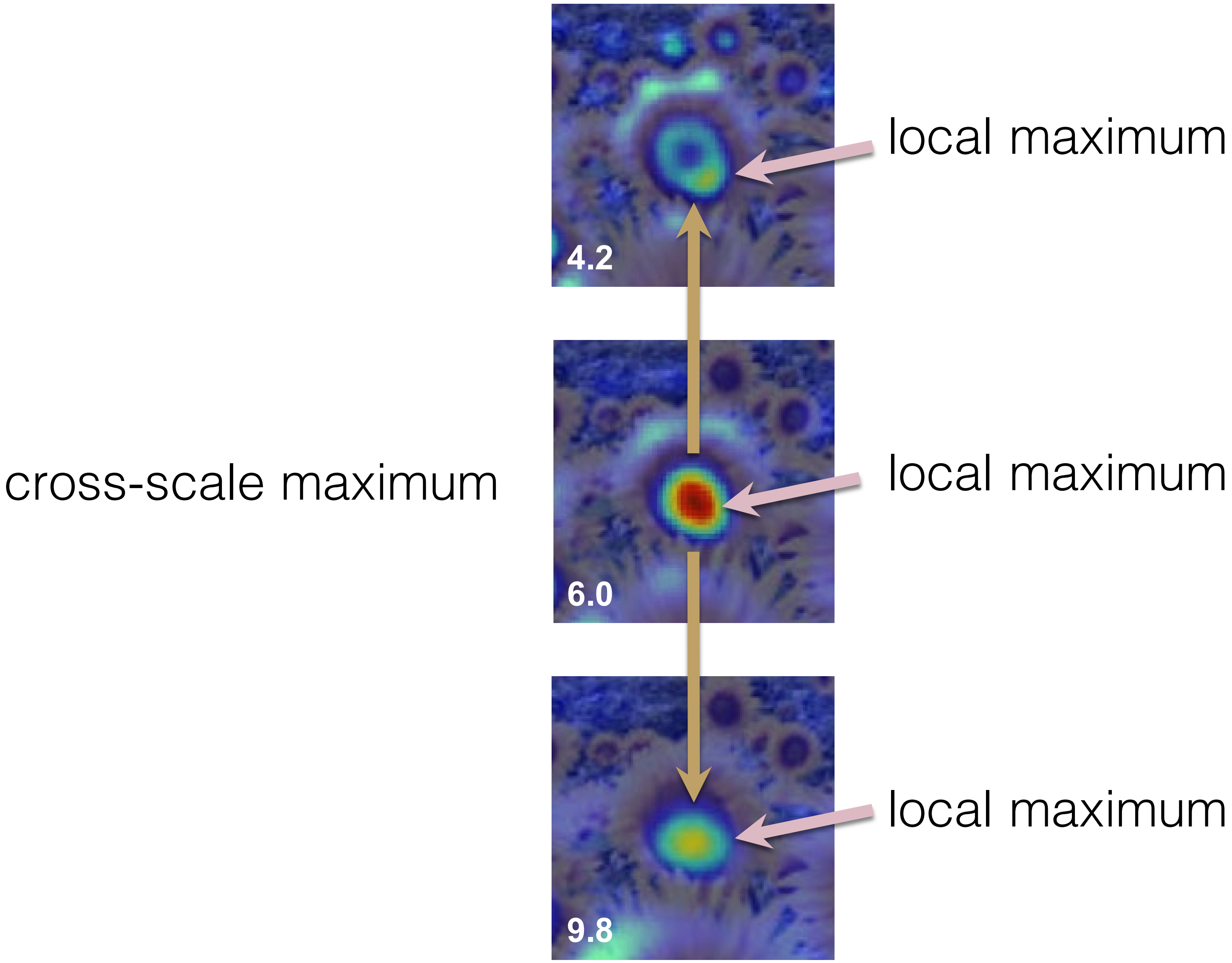


Full size

3/4 size


2.1
4.2
6.0
9.8
peak!
15.5
17.0


2.1
4.2
6.0
9.8
maximum response
15.5
17.0


4.2
local maximum
cross-scale maximum
6.0
local maximum
9.8
local maximum

Multi-Scale 2D Blob Detector

Implementation

```
For each level of the Gaussian Pyramid:
```

- `Compute feature response`
- `If` *`local maximum`* `AND` *`cross-scale`*
    - `Save location and scale of feature` (x, y, s)

We have detected corners and blobs. *But what is it useful for?*

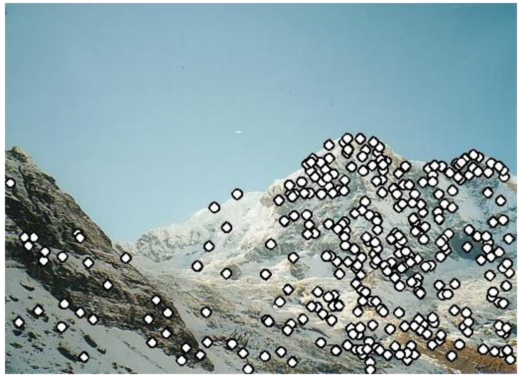
So that we can match them with related points

But how do we know that one point is similar to another point?
DESCRIPTORS

Features: Main Components

1. **DETECTION**

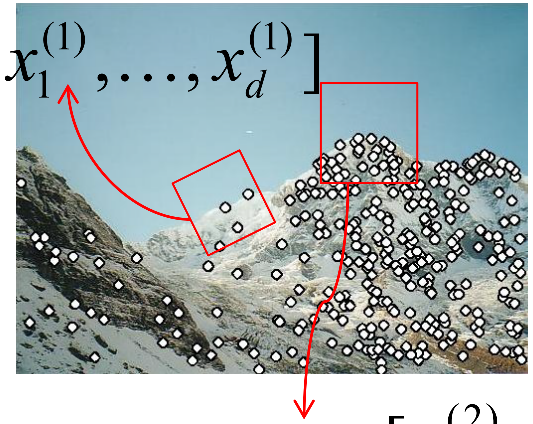
   Identify "interest points"



2. **DESCRIPTION**

   Extract "feature descriptor" vectors surrounding each interest point

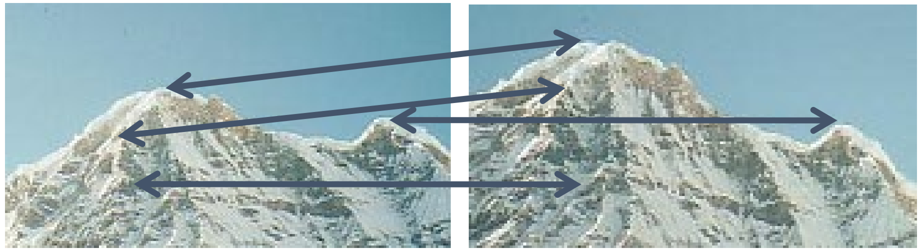
$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$



$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

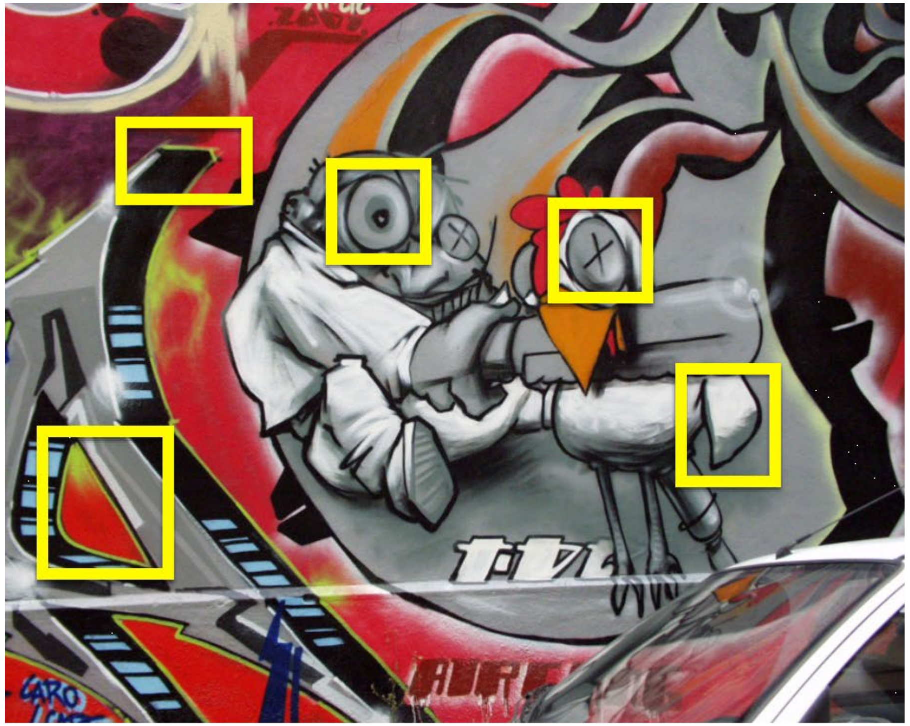
3. **MATCHING**

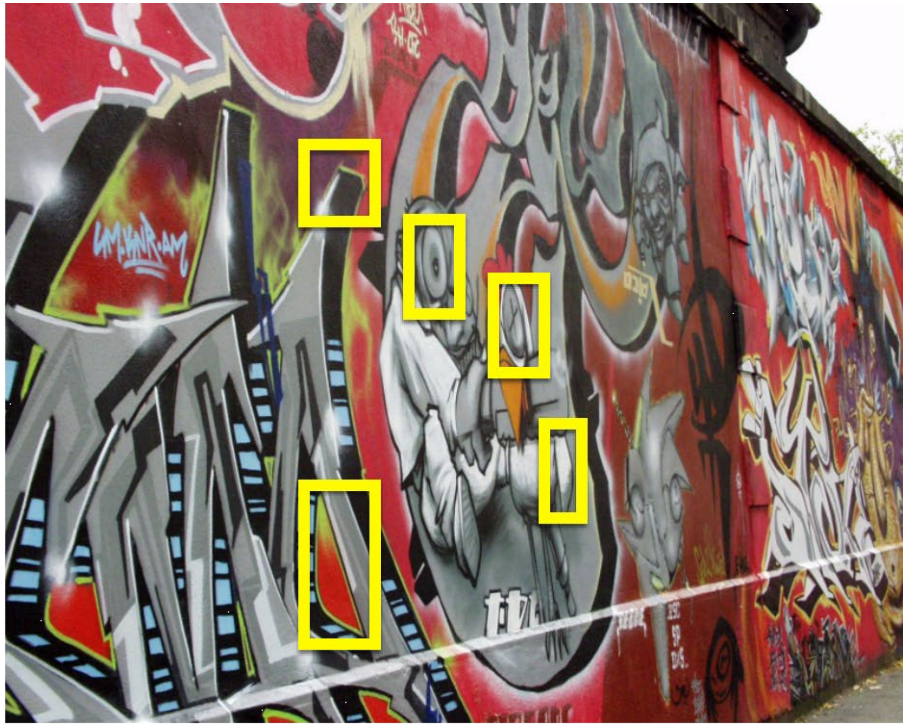
   Determine correspondence between descriptors in two views



Slide Credit: Kristen Grauman

Feature Description





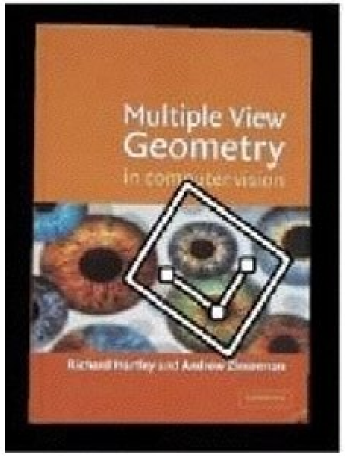
If we know where the good features are,
how do we match them?


How do we describe an image patch?
Patches with similar content should have similar descriptors.

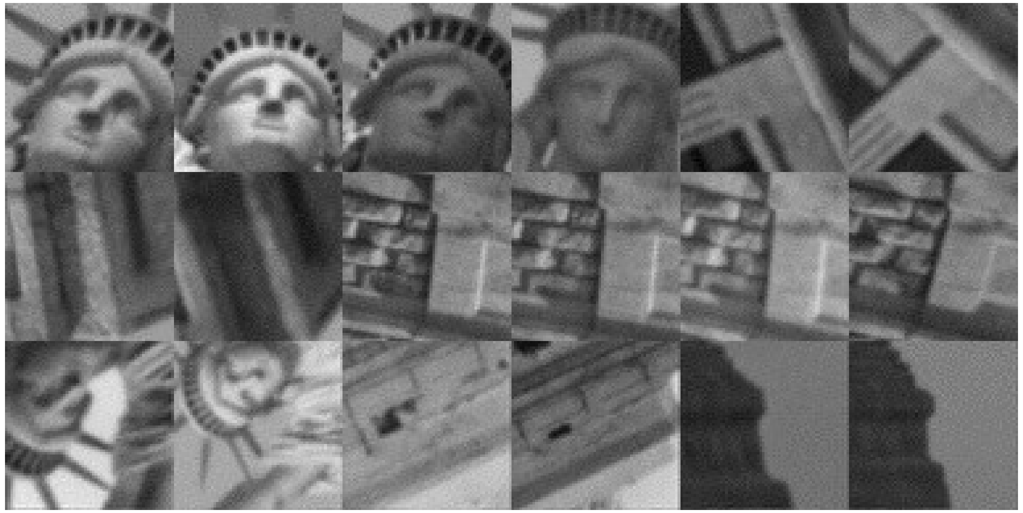
Challenges with Designing A Feature Descriptor

Photometric transformations

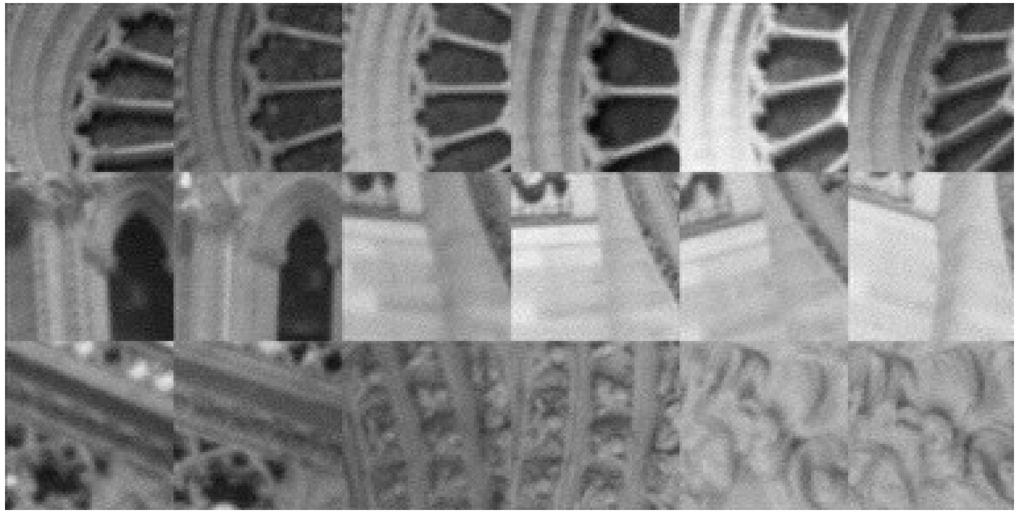
Geometric transformations



objects will appear at different scales,
translation and rotation



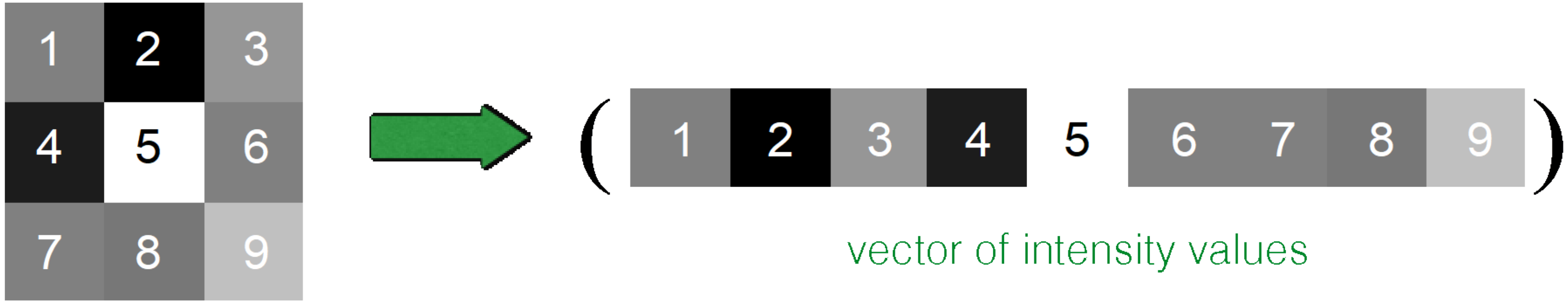
What is the best descriptor for an image feature?



Idea 1: Image Patches

Image patch

Just use the pixel values of the patch

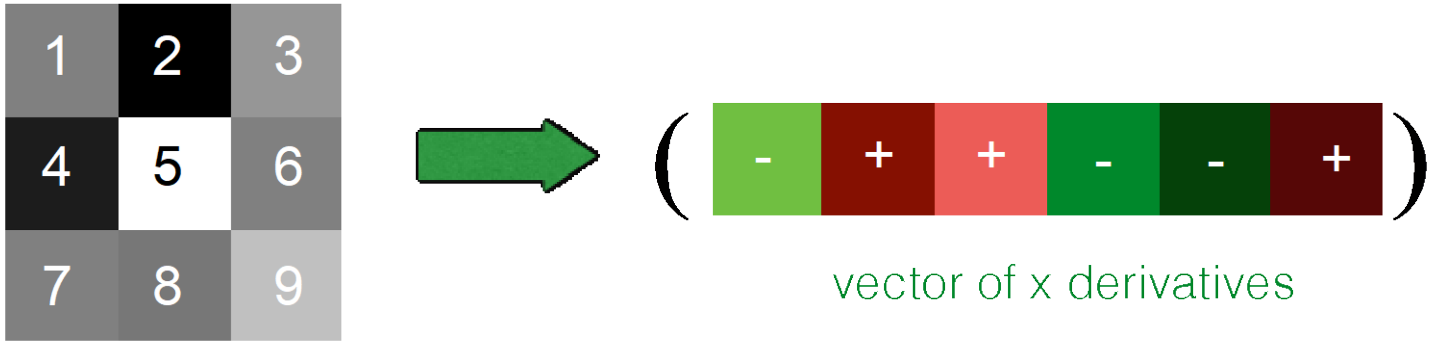


Perfectly fine if geometry and appearance is unchanged

(a.k.a. template matching)

Image gradients

Use pixel differences



vector of x derivatives

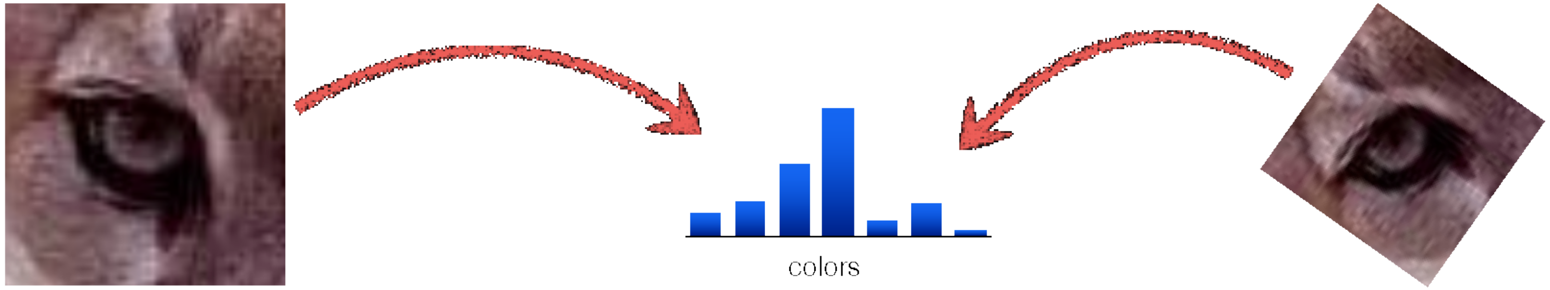
'binary descriptor'

Feature is invariant to absolute intensity values

Idea 3: Color Histogram

Color histogram

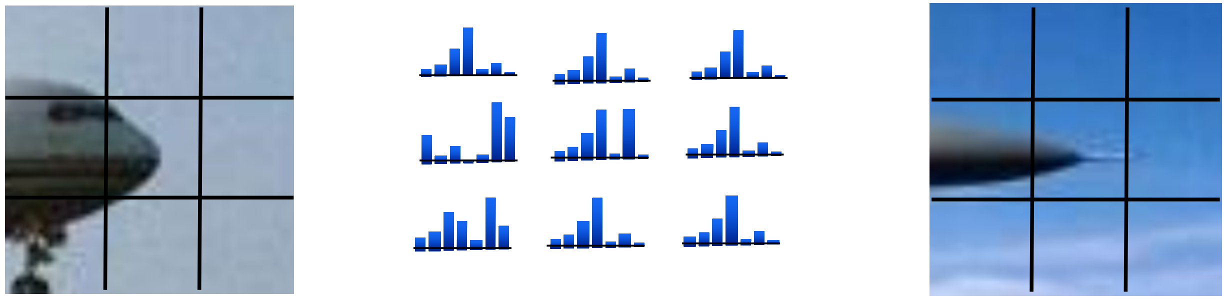
Count the colors in the image using a histogram



Invariant to changes in scale and rotation

Spatial histograms

Compute histograms over spatial ‘cells’

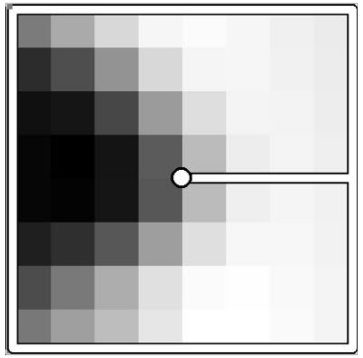


Retains rough spatial layout

Some invariance to deformations

Orientation normalization

Use the dominant image gradient direction to normalize the orientation of the patch



save the orientation angle θ along with (x, y, s)

Many more feature detectors

- MOPS (Multi-Scale Oriented Patches)
- Haar-Wavelet filterbank
- GIST features (uses Gabor filterbank)
- Textons
- HOG (Histogram of Oriented Gradients)
- SURF (Speeded-Up Robust Features)
- BRIEF (Binary Robust Independent Elementary Features)

Invariance vs. Discriminability

- Invariance:
  - Descriptor shouldn't change even if image is transformed
- Discriminability:
  - Descriptor should be highly unique for each point

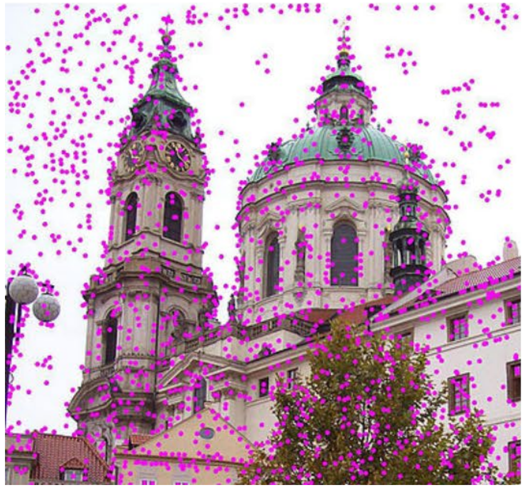
Invariant descriptors

- We looked at invariant / equivariant **detectors**

- Most feature descriptors are also designed to be invariant to:
  - Translation, 2D rotation, scale

- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transforms (some are fully affine invariant)
  - Limited illumination/contrast changes

Main (classical) feature used: SIFT



SIFT

(Scale Invariant Feature Transform)

SIFT

(Scale Invariant Feature Transform)

SIFT describes both a **detector** and **descriptor**

1. Multi-scale extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

Some history of SIFT

- The SIFT paper by David Lowe was rejected multiple times

David Lowe relates the story:

> *I did submit papers on earlier versions of SIFT to both ICCV and CVPR (around 1997/98) and both were rejected. I then added more of a systems flavor and the paper was published at ICCV 1999, but just as a poster. By then I had decided the computer vision community was not interested, so I applied for a patent and intended to promote it just for industrial applications.*
>
> *Another recent example is Rob Fergus's tiny images paper, which never did appear in a conference, but already has had a strong impact. I'm sure there are hundreds of other examples.*

Source: http://yann.lecun.com/ex/pamphlets/publishing-models.html

- SIFT went on to become the most highly cited paper in all of engineering sciences in 2005.

Scale Invariant Feature Transform

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations
- Shift the bins so that the biggest one is first

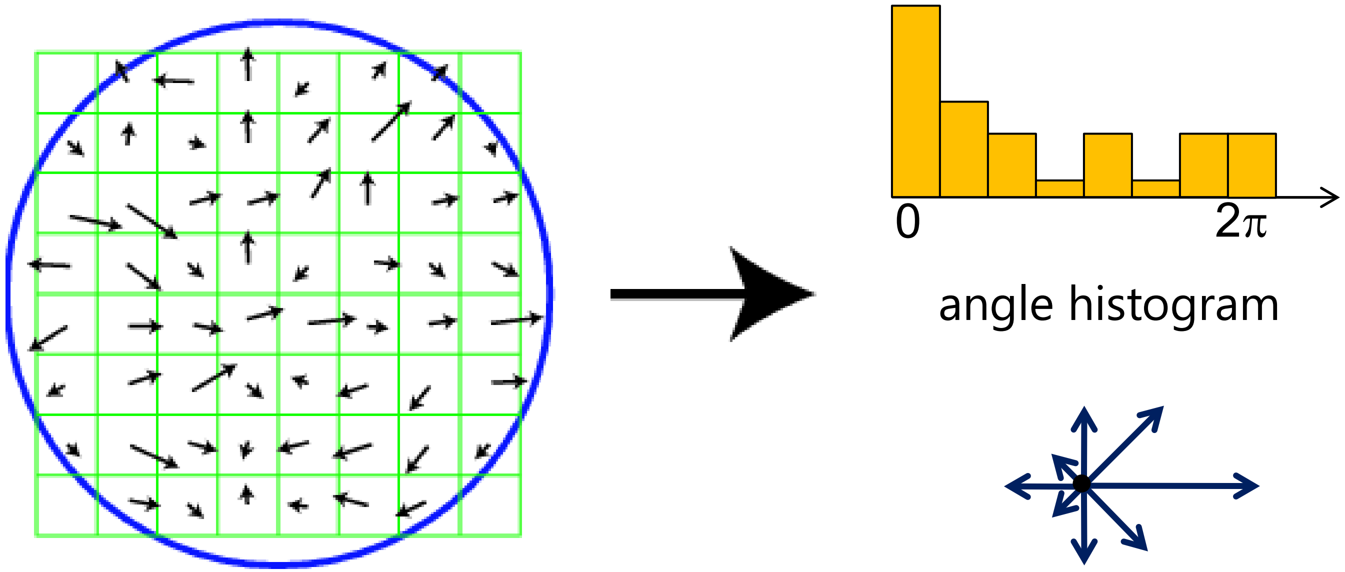
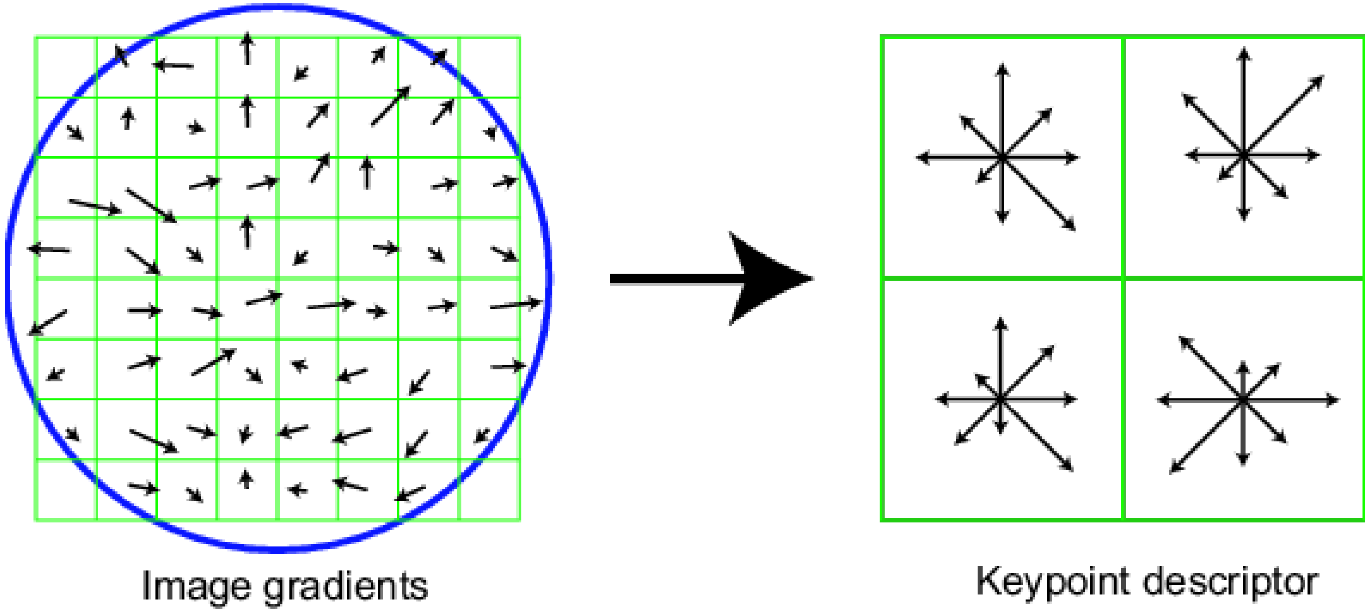


Image gradients

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

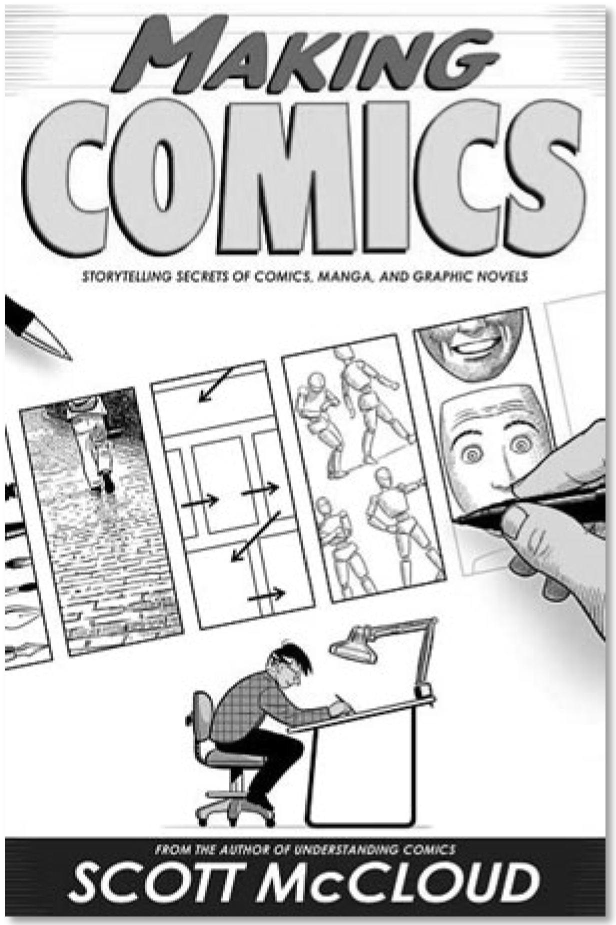


Properties of SIFT

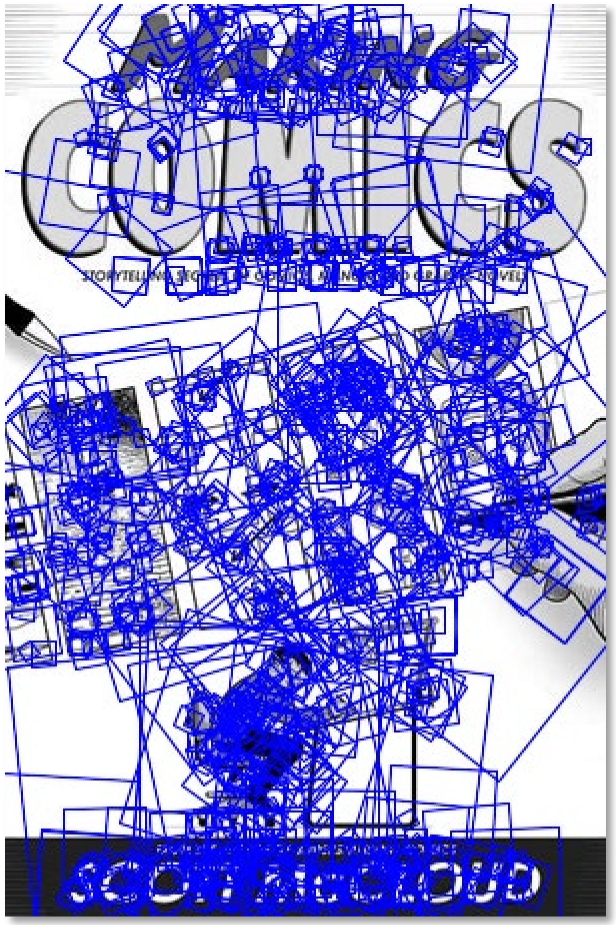
Extraordinarily robust matching technique

- Can handle changes in viewpoint (up to about 60 degree out of plane rotation)
- Can handle significant changes in illumination (sometimes even day vs. night (below))
- Pretty fast—hard to make real-time, but can run in <1s for moderate image sizes
- Lots of code available

SIFT Example

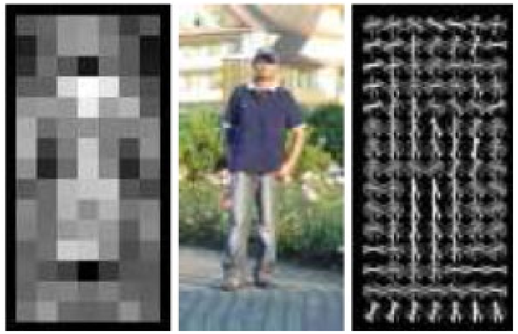


sift



868 SIFT features

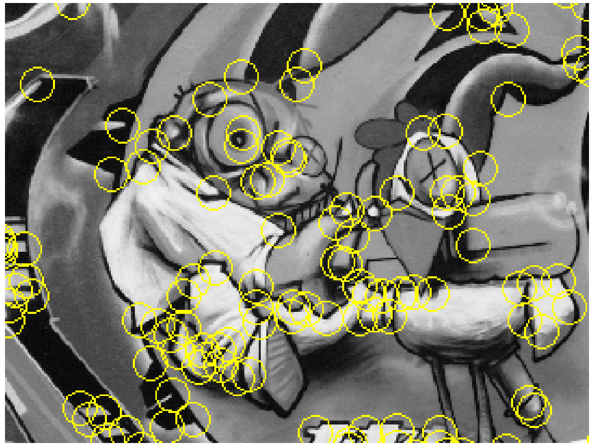
Other descriptors



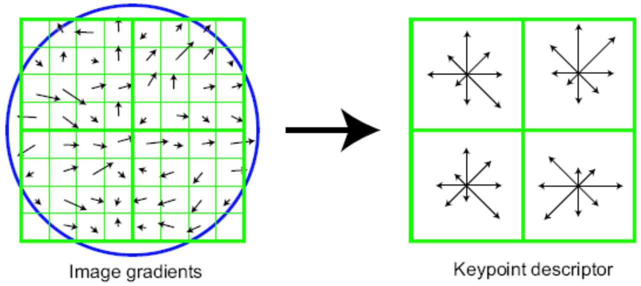
- HOG: Histogram of Gradients (HOG)
  - Dalal/Triggs
  - Sliding window, pedestrian detection
- FREAK: Fast Retina Keypoint
  - Perceptually motivated
  - Can run in real-time; used in Visual SLAM on-device
- LIFT: Learned Invariant Feature Transform
  - Learned via deep learning – along with many other recent features
    https://arxiv.org/abs/1603.09114

Summary

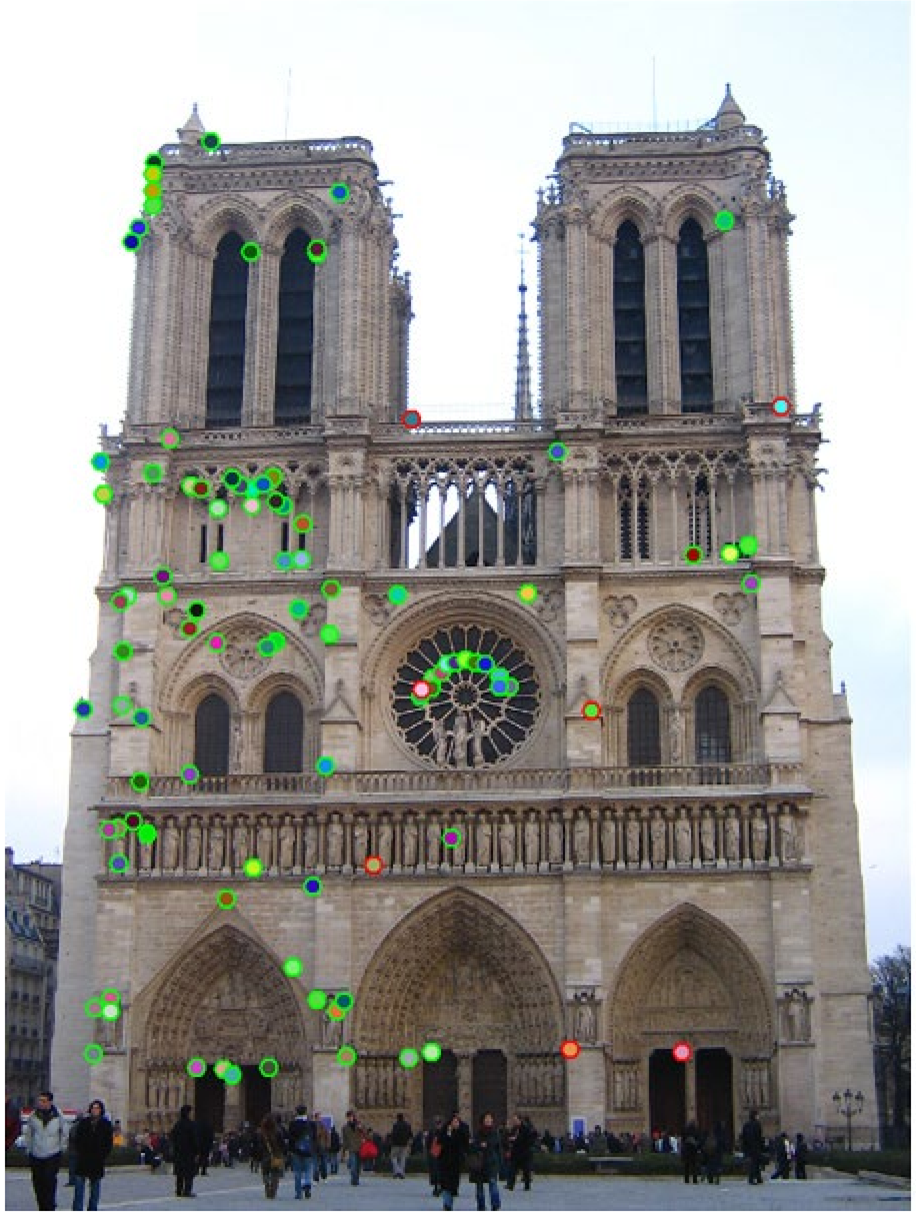
- Keypoint detection: repeatable and distinctive
  - Corners, blobs
  - Harris, DoG

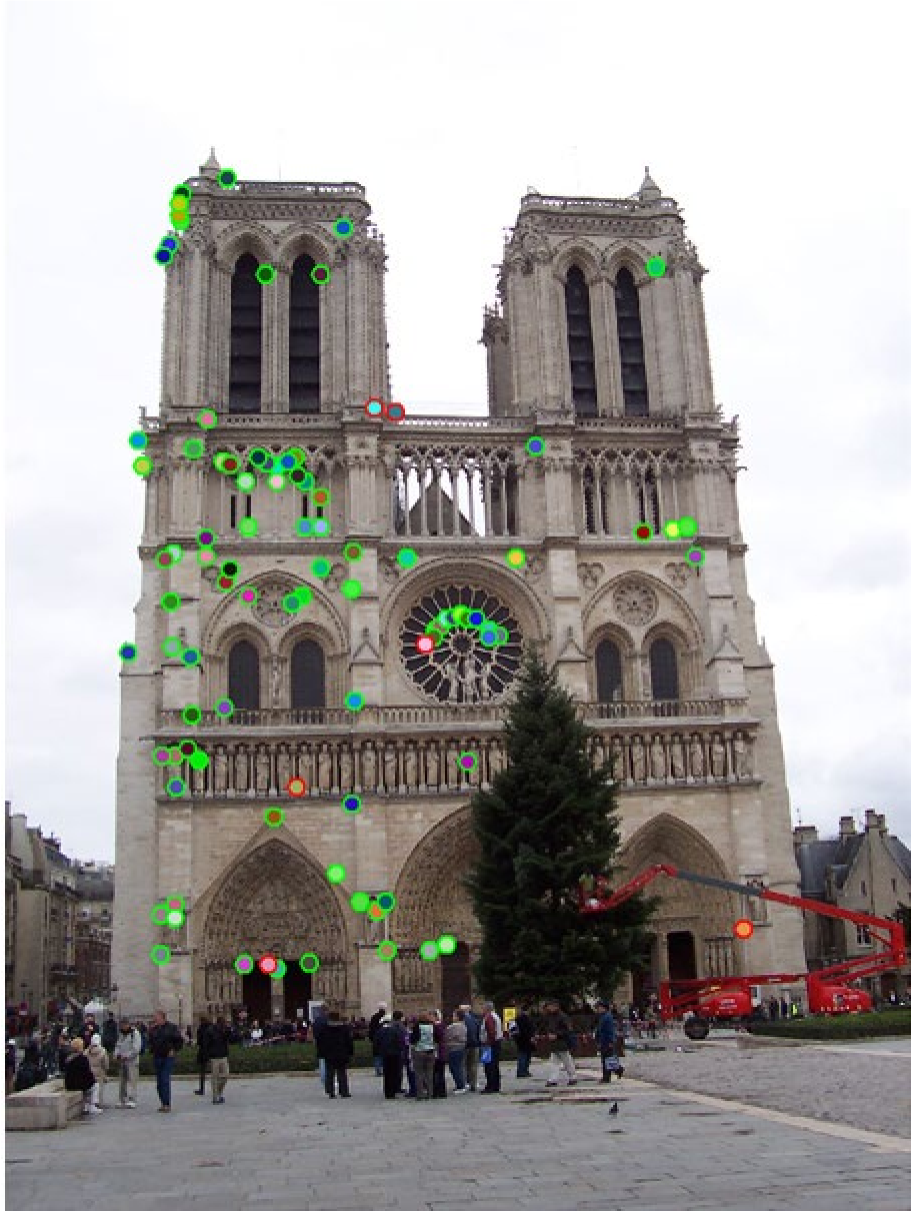


- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT and variants are typically good for stitching and recognition
  - But, need not stick to one



Which features match?





Feature matching

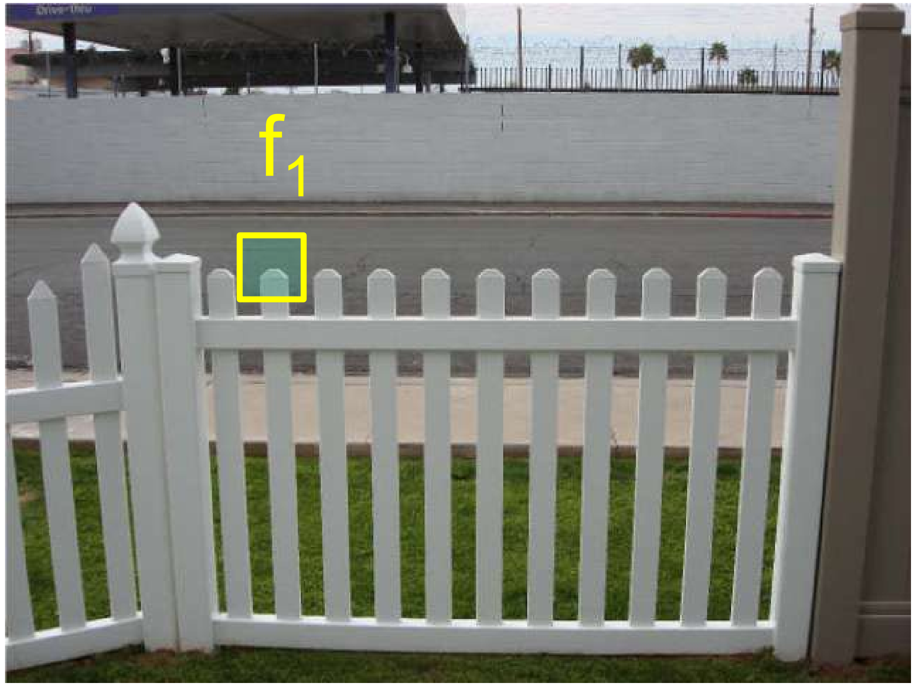
Given a feature in I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

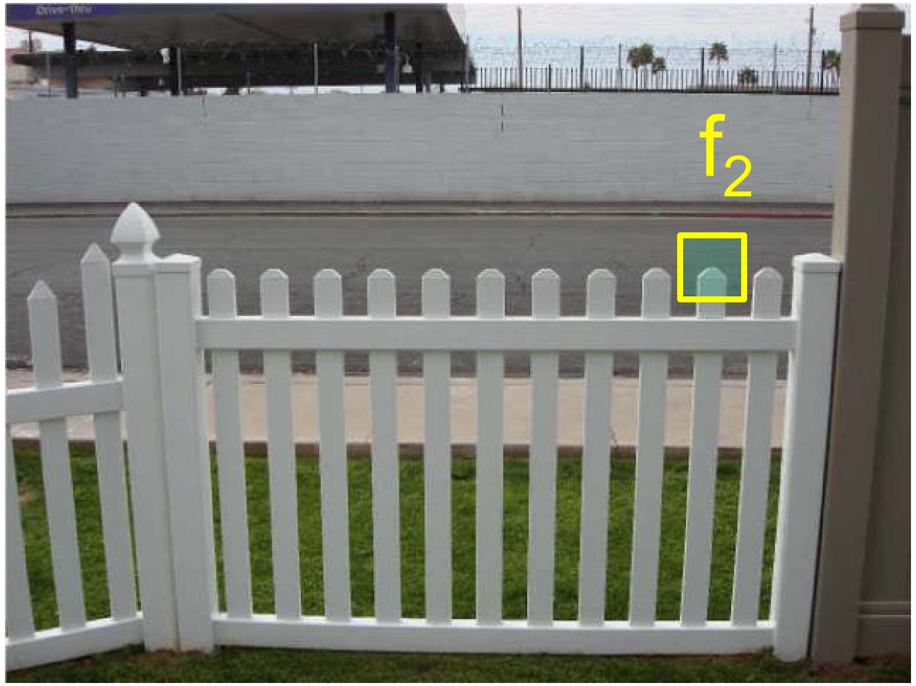
Feature distance

How to define the difference between two features f_1, f_2?

- Simple approach: L_2 distance, $\| f_1 - f_2 \|$
- can give small distances for ambiguous (incorrect) matches



I_1

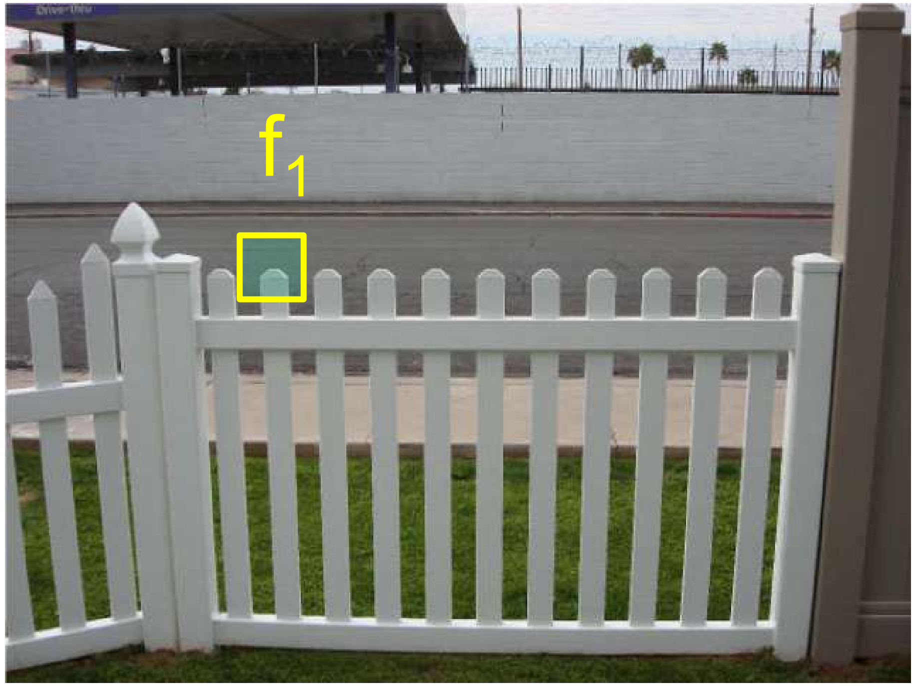


I_2

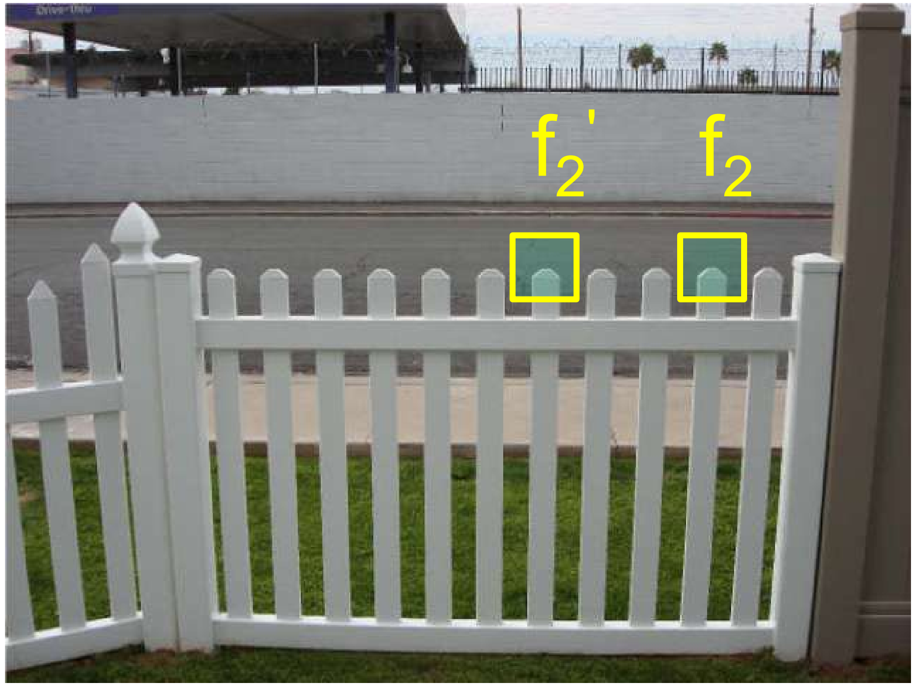
Feature distance

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = $\| f_1 - f_2 \| / \| f_1 - f_2' \|$
    - f_2 is the best SSD match to f_1 in I_2
    - f_2' is the 2^{nd} best SSD match to f_1 in I_2
    - gives large values for ambiguous matches

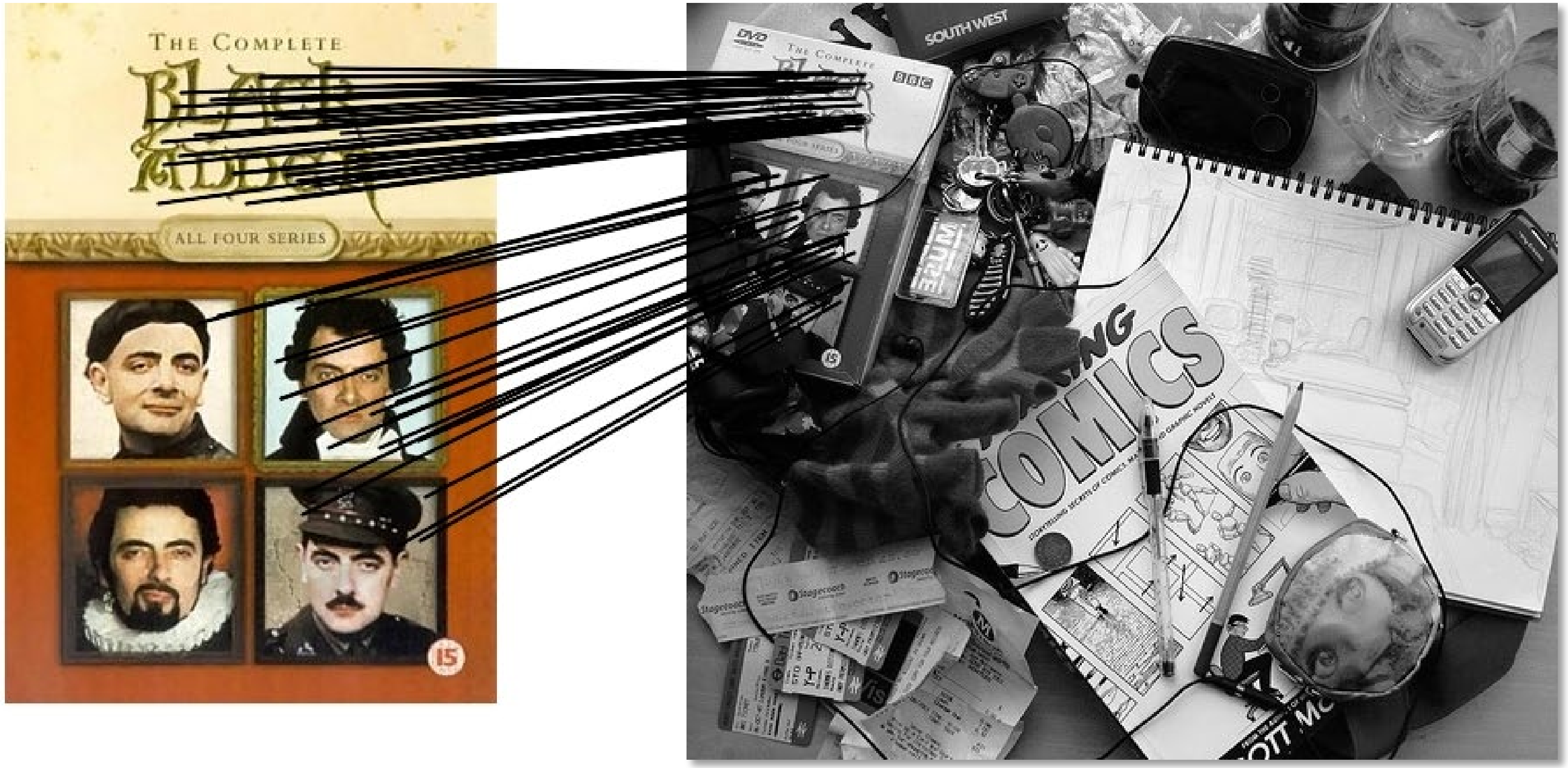


I_1



I_2

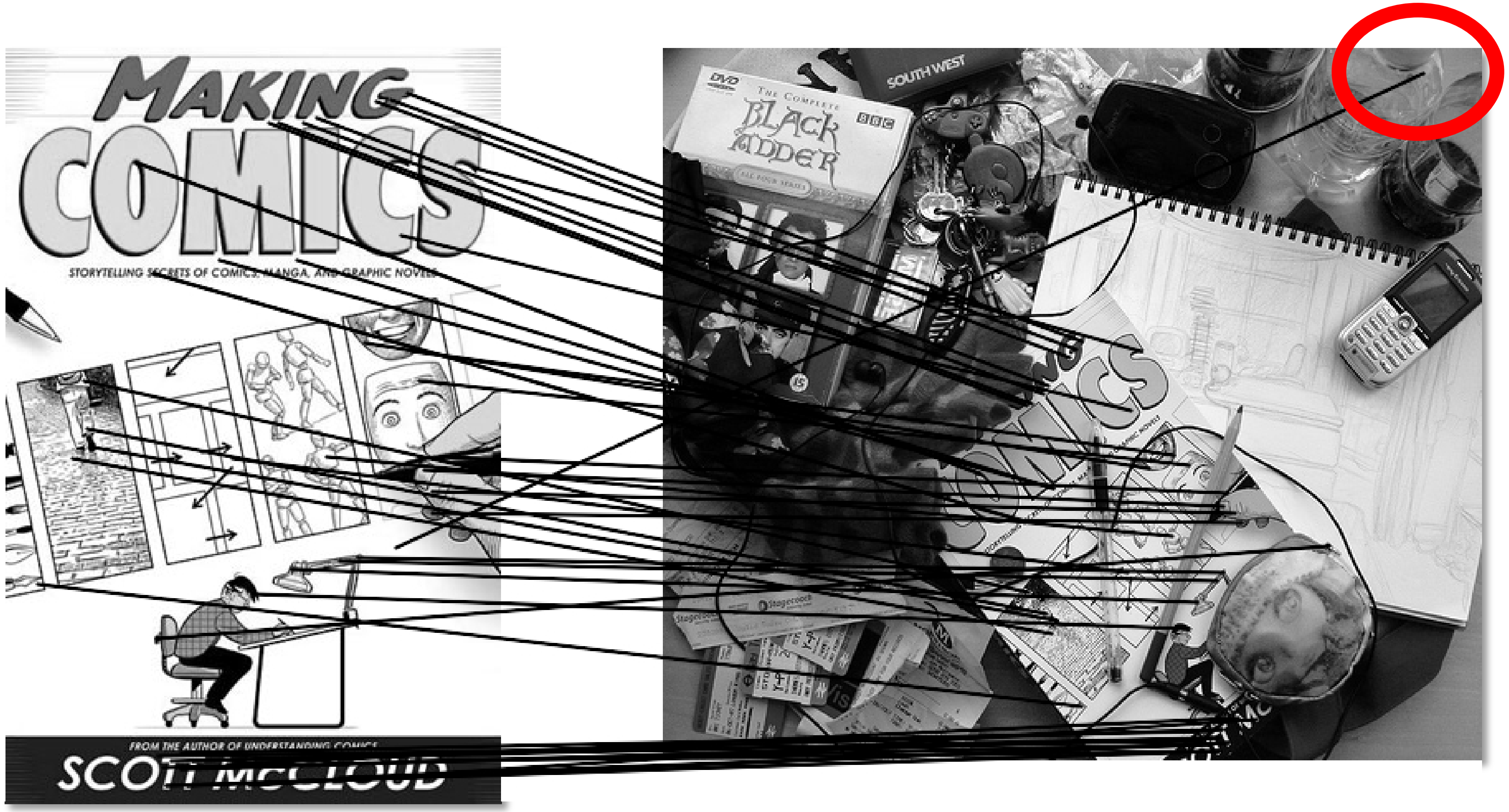
Feature matching example



58 matches (thresholded by ratio score)

Feature matching example

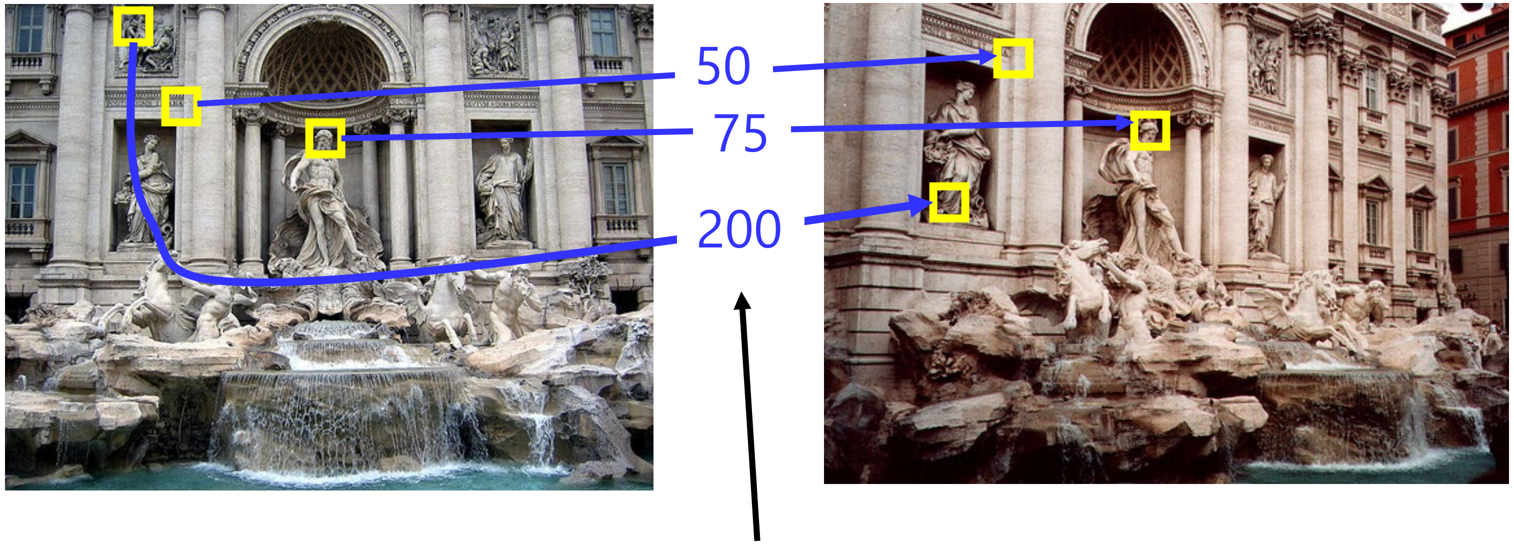
We'll deal with **outliers** later



51 matches (thresholded by ratio score)

Evaluating the results

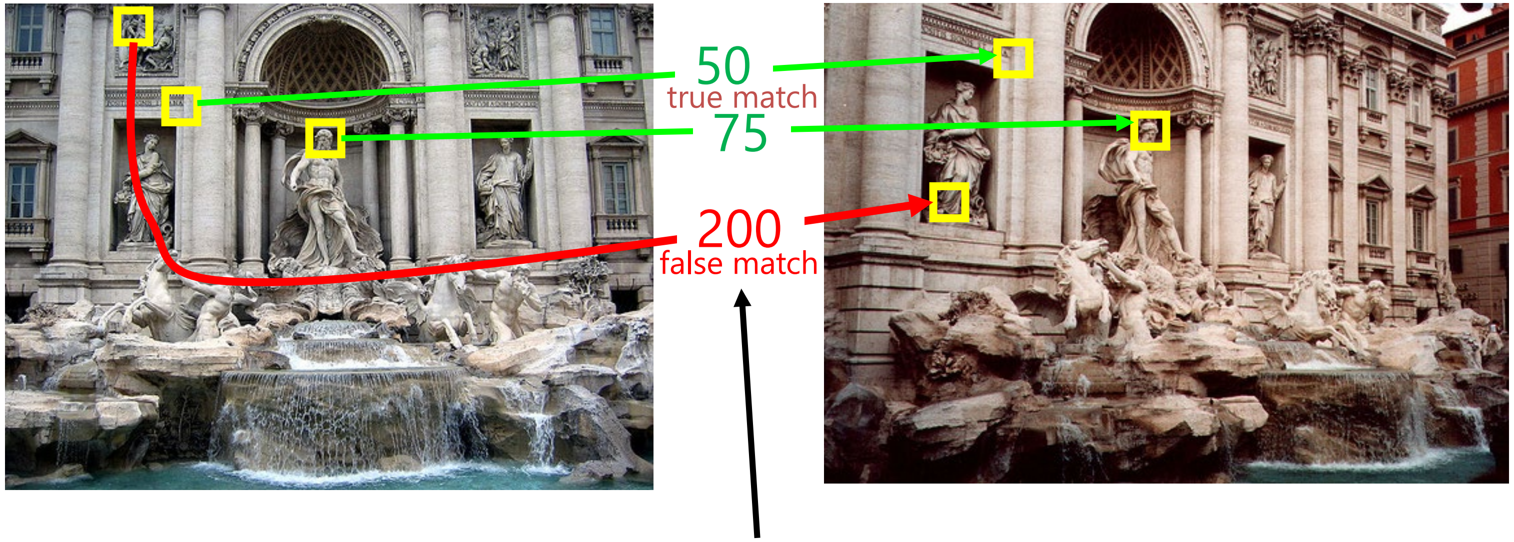
How can we measure the performance of a feature matcher?



feature distance

True/false positives

How can we measure the performance of a feature matcher?



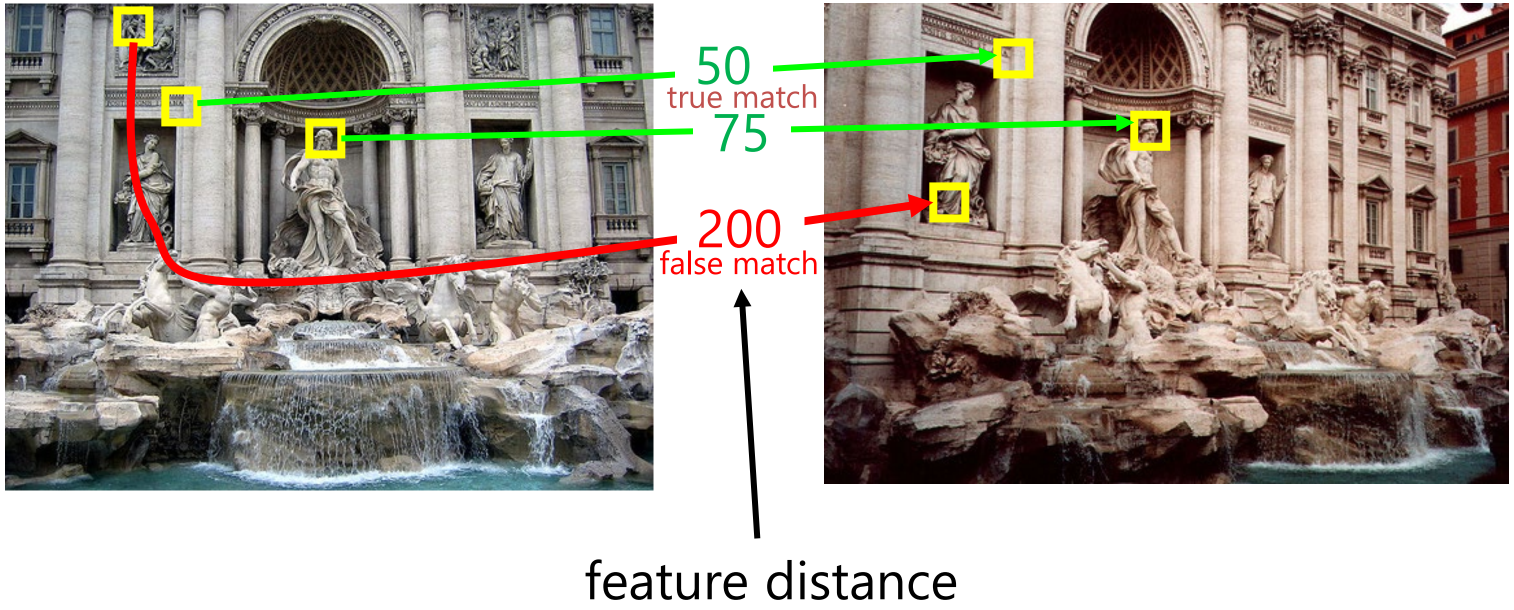
feature distance

The distance threshold affects performance

- True positives = # of detected matches that survive the threshold that are correct
- False positives = # of detected matches that survive the threshold that are incorrect

True/false positives

How can we measure the performance of a feature matcher?



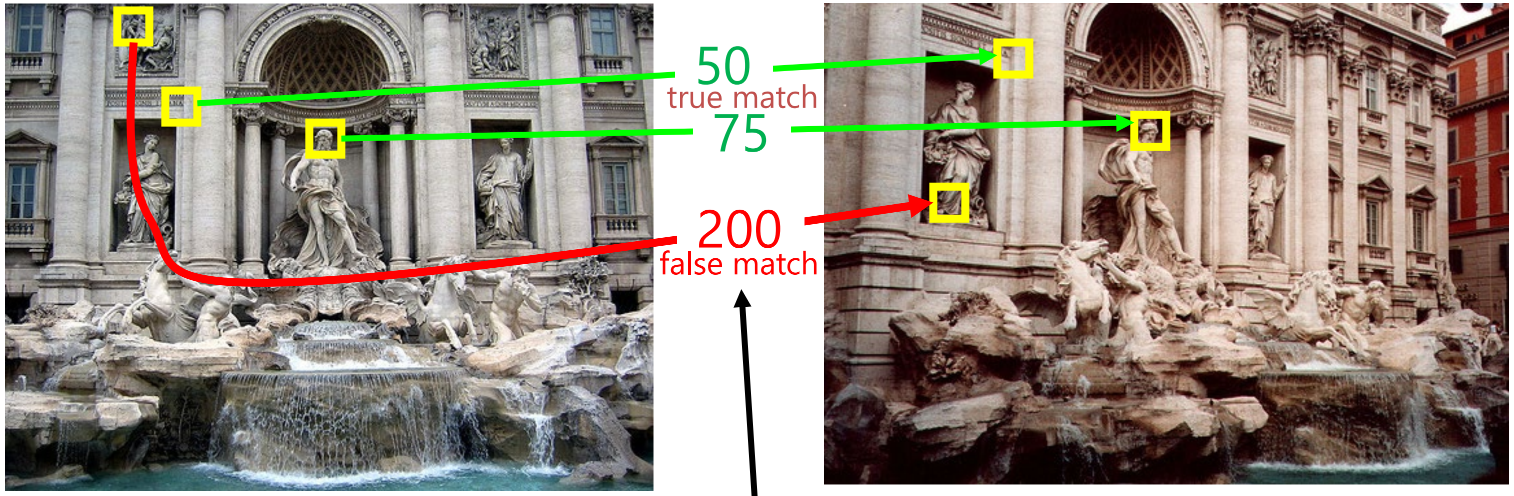
Suppose we want to **maximize true positives**.

How do we set the threshold?

(Note: we keep all matches with distance below the threshold.)

True/false positives

How can we measure the performance of a feature matcher?



feature distance

Suppose we want to **minimize false positives**.

How do we set the threshold?

(Note: we keep all matches with distance below the threshold.)

Example

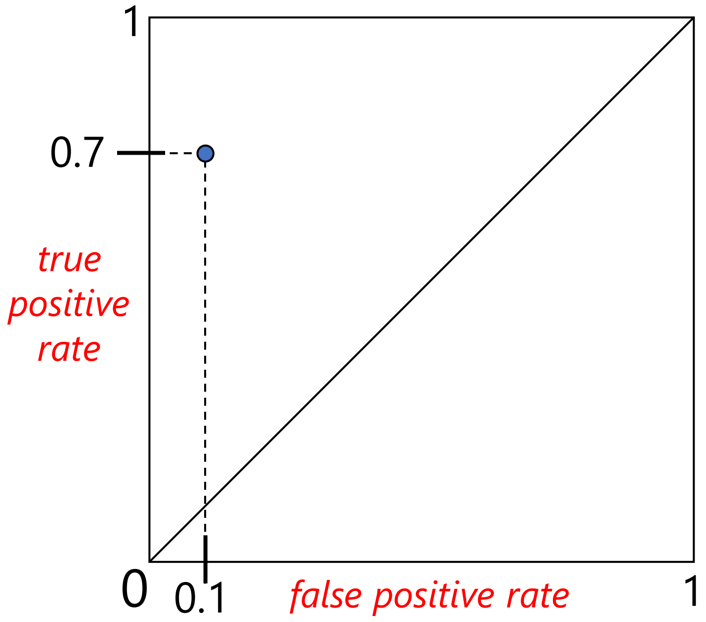
- Suppose our matcher computes 1,000 matches between two images
  - 800 are correct matches, 200 are incorrect (according to an oracle that gives us ground truth matches)
  - A given threshold (e.g., ratio distance = 0.6) gives us 600 correct matches and 100 incorrect matches that survive the threshold
  - True positive rate = 600 / 800 = ¾
  - False positive rate = 100 / 200 = ½

Evaluating the results

How can we measure the performance of a feature matcher?

$\frac{\text{\# true positives surviving threshold}}{\text{\# total correct matches (positives)}}$

recall

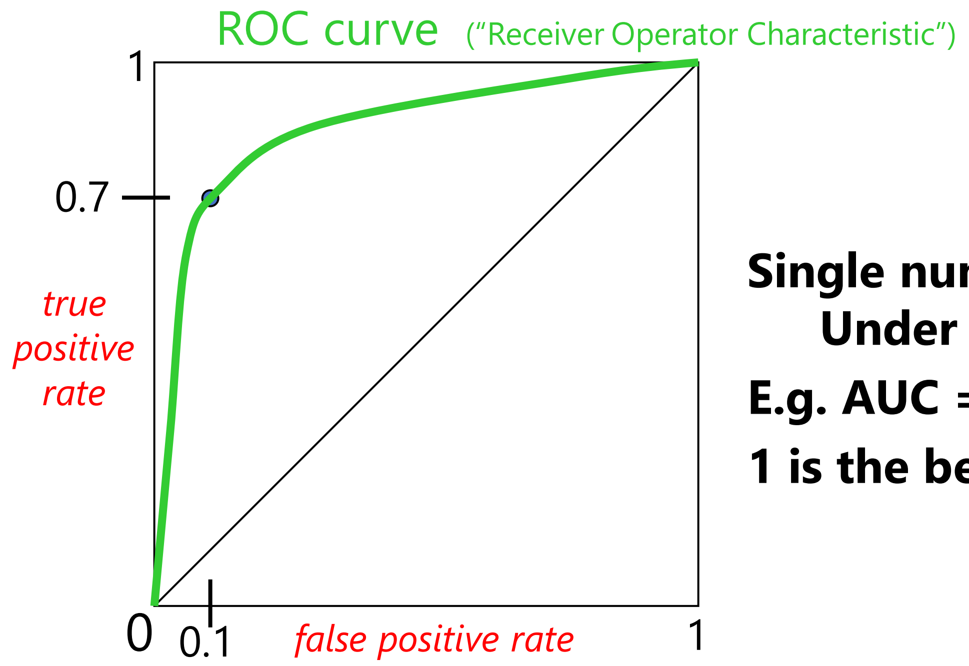


$\frac{\text{\# false positives surviving threshold}}{\text{\# total incorrect matches (negatives)}}$

1 - *specificity*

Evaluating the results

How can we measure the performance of a feature matcher?



$$\frac{\text{\# true positives surviving threshold}}{\text{\# total correct matches (positives)}}$$

recall

$$\frac{\text{\# false positives surviving threshold}}{\text{\# total incorrect matches (negatives)}}$$

1 - *specificity*

Single number: Area Under the Curve (AUC)

E.g. AUC = 0.87

1 is the best

ROC curves – summary

- By thresholding the match distances at different thresholds, we can generate sets of matches with different true/false positive rates
- ROC curve is generated by computing rates at a set of threshold values swept through the full range of possible threshold
- Area under the ROC curve (AUC) summarizes the performance of a feature pipeline (higher AUC is better)

Feature Matching is Useful for …

Object instance recognition

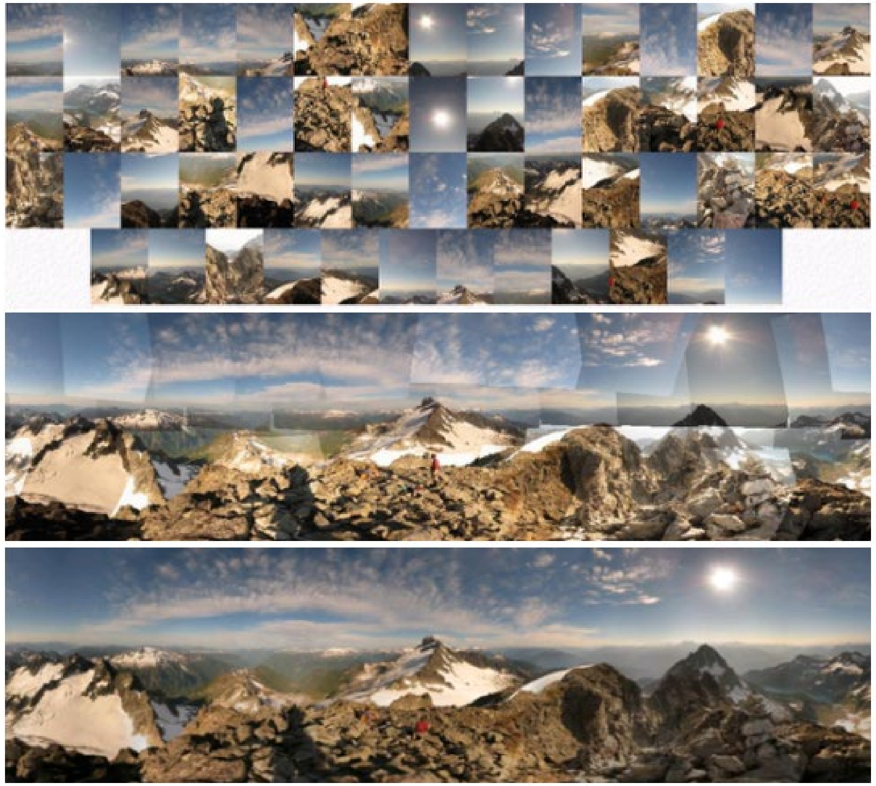
Schmid and Mohr 1997

Sivic and Zisserman, 2003

Rothganger et al. 2003

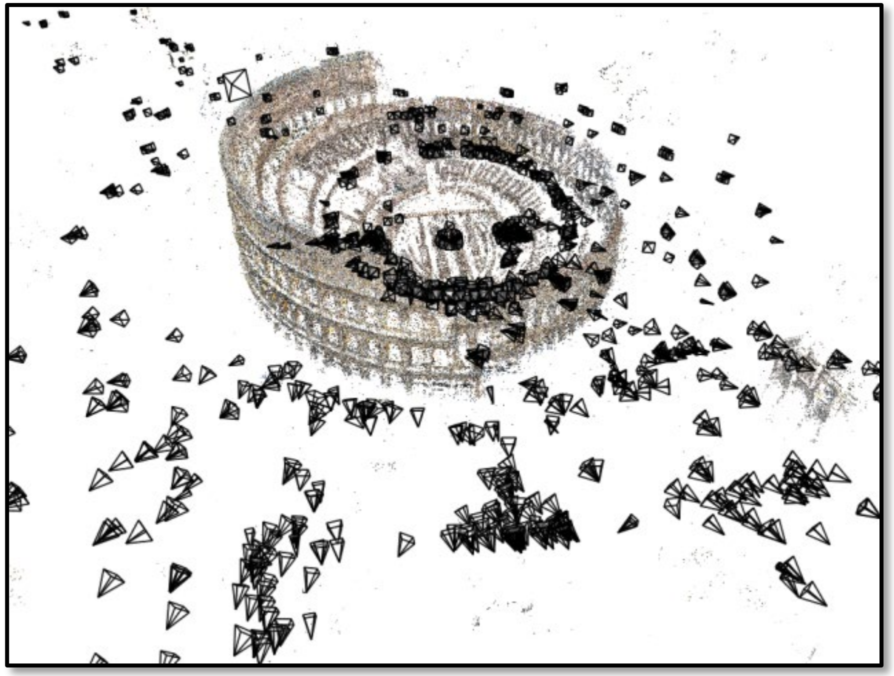
Lowe 2002

Image mosaicing



3D Reconstruction

Internet Photos ("Colosseum")



Reconstructed 3D cameras and points

Augmented Reality